The "last mile" of data handling: Fermilab's IFDH tools

Adam L Lyon and Marc W Mengel
Fermilab, P.O. Box 500, MS 369, Batavia IL 60510
E-mail: lyon@fnal.gov
E-mail: mengel@fnal.gov

Abstract. IFDH (Intensity Frontier Data Handling), is a suite of tools for data movement tasks for Fermilab experiments and is an important part of the FIFE (Fabric for Frontier Experiments) initiative described at this conference. IFDH encompasses moving input data from caches or storage elements to compute nodes (the "last mile" of data movement) and moving output data potentially to those caches as part of the journey back to the user. IFDH also involves throttling and locking to ensure that large numbers of jobs do not cause data movement bottlenecks. IFDH is realized as an easy to use layer that users call in their job scripts (e.g. "ifdh cp"), hiding the low level data movement tools. One advantage of this layer is that the underlying low level tools can be selected or changed without the need for the user to alter their scripts. Logging and performance monitoring can also be added easily. This system will be presented in detail as well as its impact on the ease of data handling at Fermilab experiments.

1. Experiment Software Time-line

The way experiment software is used changes as the experiment progresses: from design, to detailed modeling, to data analysis. A detailed timline of this follows, and then a discussion of how our "ifdh" system attempts to ease the transitions in this process. Intensity Frontier experiments at Fermilab seem to be following a fairly common pattern in their use of software and storage:

- They start off with high rates of software change and small sets of data files of simulated data, and they run that software over that data (and regenerate that data) onto our central BlueArc NFS Filer, and all their scripts and tools run directly off of the NFS storage.
- As the software stabilizes, they begin to generate larger sets of simulated data, and then as the detectors and other equipment come on line, they start recording actual data. Now they begin using local batch resources for generation and analysis, and have to learn to stage data in and out of the central NFS storage with throttling tools, so they don't overload the NFS service. They also begin building deep hierarchies of directories in the file store to organize and keep track of their data, and building "playlists" of data files to analyze.
- As their data grows, they come to the realization that they cannot keep requesting more space on the central NFS filer anymore, and they really need to migrate to using hierarchical storage, and to using a meta-data catalog to keep track of their files.

The design of our "ifdh" layer attempts to ease the transitions between these differing modes of software operations. When experiments make the first transition from running interactively off of the NFS filer to running batch jobs, they have a simple transition to make from reading and writing directly to files on the NFS filer, to staging it locally. We can see how this looks in a sample job shell script sample as it evolves over time.

The experimenters initially convert from:

```
while read sourcefile
do
    framework_exe -c config $sourcefile $destfile
done < playlist
```

to

```
while read sourcefile
do
    ifdh cp $sourcefile localsource
    framework_exe -c config localsource localdest
    ifdh cp localdest $destfile
done < playlist
```

That is, they simply use "ifdh" to get the files to and from NFS, as if with plain Unix "cp", and it throttles the copies so the file server isn't overloaded, and they go along as usual.

Their next big transition is when they start using the SAM file catalog instead of playlist files. As far as the scripts they run, this is again a simple transition using "ifdh" – instead of reading a playlist, they get files from a SAM "project", which deals out files from a dataset to one or more running jobs:

```
while uri=‘ifdh getNextFile $projectUrl‘ && [ -n “$uri” ]
do
    localsource=‘ifdh fetchInput $uri‘
    framework_exe -c config $localsource localdest
    ifdh cp localdest $destfile
done
```

and now we have a job script fragment that can use multiple jobs to process a data-set defined in the SAM data catalog. In some cases it is even simpler, because the getNextFile loop can be loaded into the Art Framework used by many of our Intensity Frontier experiments. (Note that this is actually a slightly simplified example; a full job script example is at https://cdcvs.fnal.gov/redmine/projects/ifdhc/repository/revisions/master/entry/demo.sh)

As these jobs move out onto the Open Science Grid, they now do not need to change these scripts at all – the ifdh fetchInput and ifdh cp utilities know how to use Grid software utilities to transfer files to and from Fermilab, without changing the user's scripts. The "ifdh" utility can pick the best mechanism for transferring data back to Fermilab by looking around at the environment, or users can explicitly choose by passing environment variables into their jobs.

2. ART Framework Integration

For our experimenters using the ART Framework; there is also an ifdh_art set of libraries, which lets users specify a few command-line arguments to transparently switch from play-list-file style operation to working from SAM datasets with ifdh file transfers.

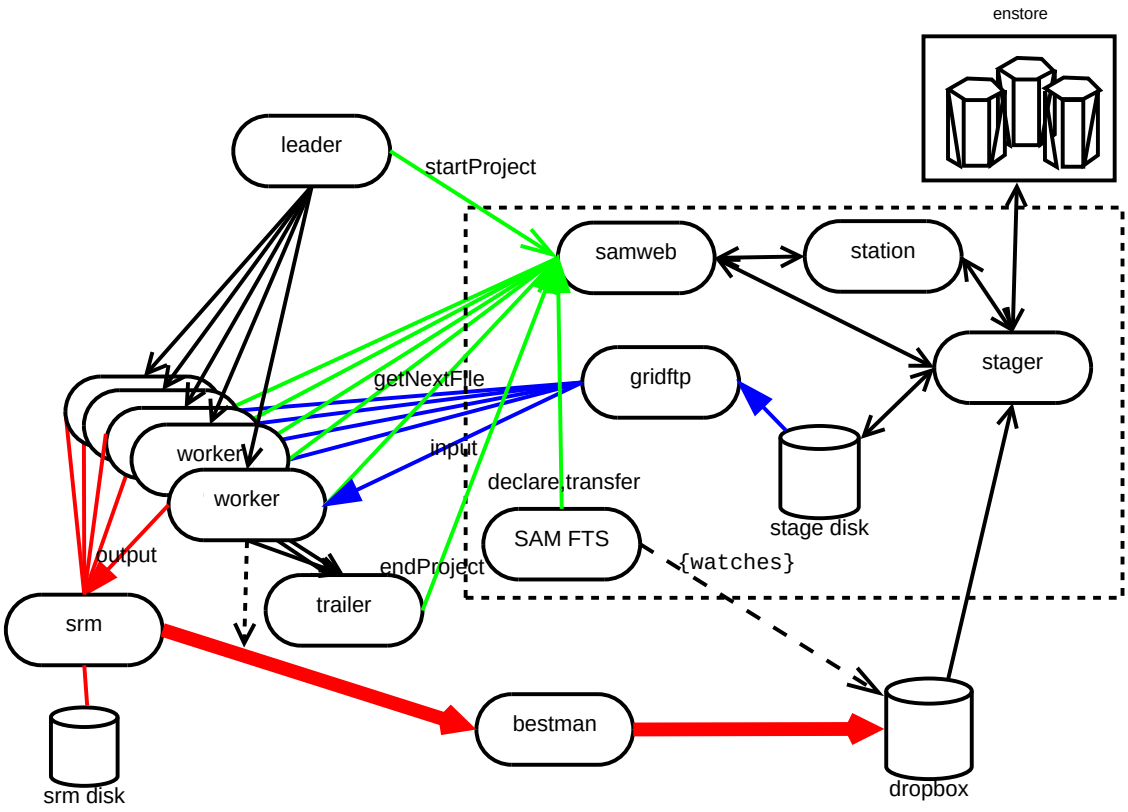
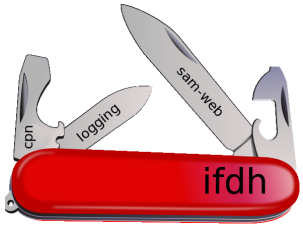


Figure 1. Data flow of analysis jobs using SAM
SAM requests in green,input data in blue,output data in red

3. Implementation Hiding

The simple invocation of "ifdh cp" can hide anything from a simple Unix cp to staging a file to a Grid storage element, and then third-party copying the file back to Fermilab. When we encourage our experiments to use this layer, we give ourselves the flexibility to change how data is transferred, without the user having to change their software or scripts. Figure 1 shows the overall system, where users jobs are run in a Condor DAG (black arrows), with a leader job starting a SAM "project", SAM staging data in from hierarchical storage as needed, each job making getNextFile calls and using ifdh to copy the input files locally, and the ifdh cp to send the data back to Fermilab is done by staging the files to an SRM near the job, and one job slot taking on the role of copying back files via third-party-copy to Fermilab. The ifdh utility is providing all of the colored lines in this diagram.



4. Swiss Army Knife design

The "ifdh" utility is designed to be the software equivalent of a Swiss Army Knife – a whole set of useful tools in a single executable; it has a lightweight client for a subset of the SAM Web API, it has the ifdh cp and ifdh mv commands for moving data, which can use gsiftp: or srm: protocol tools, or the site local CPN locking scripts for transferring data using NFS without overloading the servers. This means that all of the functions that "ifdh" handles for you are available quickly, without requiring large job tarballs or large copies via CVMFS.

5. Language Support
The ifdh utilities are available as a command line utility, a C++ shared library, and a python library, with matching calls, so that experiments can integrate it as they find appropriate.

In C++:

```
#include "ifdh.h"
ifdh i();
location = i.locateFile(base_uri, filename);
```

in Python:

```
import ifdh
i = ifdh.ifdh()
location = i.locateFile(base_uri, filename);
```

In bash:

```
location=‘ifdh locateFile $base_uri $filename‘
```

Tools included in the ifdh executable are:

- SAM dataset tools let programs define data-sets and examine what files are included in them.
- SAM project tools (start, find, getNextFile, updateFileStatus, end) let programs cooperate to process an entire dataset, and keep track of progress and completion.
- SAM locate file tools let scripts find what copies of known files are registered with SAM.
- File transfer tools (ifdh cp, mv, getInput, copyBackOutput) chooses transfer method automatically, but can be overriden supports:
  - dd w/throttling for NFS
  - srm copy wrapper
  - gridftp wrapper
  - srm with intermediate staging

6. Lightweight
The ifdh executable that provides the above tool weighs in under 1 megabyte and relies on only a handful of standard system libraries.

7. Logging
The ifdh utility also uses RFC5426-style messaging to log activity over UDP packets back to an rsyslog service on our monitoring server, where we can keep track of file copies, data rates, etc., as well as having the ability to forward log messages (i.e. for a particular experiment) to other rsyslog servers. Users can also use “ifdh log” to log their own messages in their scripts.

8. Conclusions
A thin layer of software to abstract the “last mile” of data handling for experiment jobs is extremely useful for assisting experiments in migrating from early software development to high volume production operation, and for giving flexibility to standardize, monitor, and change how the last leg of data handling is performed, without making experimenters change their software and scripts. The “ifdh” executable/library approach is working well in practice, and has already allowed us to make experimenters’ scripts able to operate in different batch environments using differing file transfer mechanisms without changing their scripts.

FERMILAB-CONF-13-481-CD.